On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices

Vingelmann, Peter; Heide, Janus; Pedersen, Morten Videbæk; Fitzek, Frank

Published in:
Globecom. I E E E Conference and Exhibition

DOI (link to publication from Publisher):
10.1109/GLOCOM.2011.6133577

Publication date:
2011

Document Version
Early version, also known as pre-print

On-the-fly Packet Error Recovery in a Cooperative Cluster of Mobile Devices

Péter Vingelmann*†, Morten Videbæk Pedersen†, Frank H. P. Fitzek†, and Janus Heide†
*Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Hungary
†Department of Electronic Systems, Faculty of Engineering and Science, Aalborg University, Denmark

***Abstract*—This paper investigates the possibility of packet error recovery in a cooperative cluster of mobile devices. We assume that these devices receive data from a broadcast transmission on their primary network interface (e.g. LTE network), and they are using a secondary network interface (e.g. ad hoc WLAN network) to form a cooperative cluster in order to exchange missing data packets among each other. Our goal is to devise a protocol that minimizes the number of packets exchanged on the secondary network whilst maximizes the number of packet errors recovered on the primary network. Moreover, we aim to repair the packet losses on-the-fly (as the data is being received), which also imposes real-time constraints on the protocol. We propose a solution based on random linear network coding to form cooperative clusters of mobile devices to facilitate the efficient exchange of information among them. We also introduce a demo application that implements this technique on Nokia phones. Then we present our testbed and the collected measurement results in order to evaluate the performance of our protocol.**

I. Introduction

Network coding has received a lot of attention lately [1], [2], [10], [8], [12]. Researchers have shown that network coding has its clear advantages, especially for wireless and mobile multi-hop communication systems. Our prior work has been focused on the feasibility and tuning of network coding for mobile platforms [5], [14], [13]. Later those findings were confirmed by other researchers [15], so we could conclude that network coding is feasible on embedded devices, and its benefits in terms of energy consumption and bandwidth usage are realistic. Researchers in [11], [4] proposed solutions that utilize network coding in cooperative clusters in various communication scenarios.

The work in this paper is also focusing on protocol design based on network coding. In particular, we investigate the possibility of using cooperative clusters of mobile devices for packet error recovery in content distribution systems. The main architecture of these systems has not changed much in the past decade. In the mobile world, we still use a highly centralized client/server architecture, where the overlay network is providing the content, and the mobile users are merely consuming it.

The protocol proposed here takes advantage of the mobile devices themselves by forming cooperative clusters with the help of network coding. We attempt to enhance the traditional client/server communication pattern by allowing neighboring devices to communicate directly to make the content distribution more efficient.

This paper is organized as follows. Section II presents our scenario and gives an overview on network coding. Section III discusses the features of our protocol and introduces our application. Section IV presents measurement results. Future works are discussed in Section V, and the final conclusion is drawn in Section VI.

II. Scenario

In our scenario a source wants to reliably transmit the same media file or media stream to several receivers. We assume that these receivers are connected to the source via their primary network interface (e.g. 4G or LTE network). In state-of-the-art systems the source *broadcasts* the data on this interface, and it also applies some sort of coding scheme (e.g. Raptor coding) to fix the packet erasures on the receivers. Since packet losses are quite common in wireless networks, this coding may require a large overhead. Therefore we assume that the receivers are also able to communicate with each other using their secondary network interface (e.g. ad hoc WLAN). Thereby cooperative clusters can be formed of several receivers to exchange missing data packets among each other. This basic scenario is depicted in Figure 1.

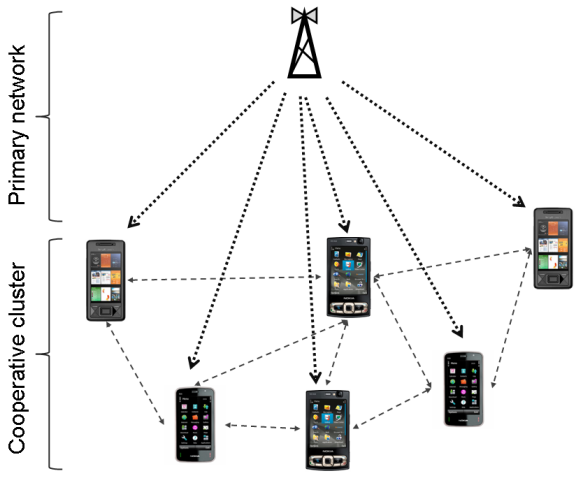


Fig. 1. A source transmitting data to multiple receivers.

The fundamental problem with broadcast packets is the frequent losses of packets in real-life wireless networks [6]. Since channel conditions are not ideal, we cannot expect that all

broadcast packets are delivered to all receivers. Packet losses must be corrected by using some sort of retransmission scheme to ensure reliability. As a naive solution, the individual nodes can request all their missing packets from the original source. The end result would be similar to the Automatic Repeat-reQuest (ARQ) mechanism used in one-to-one transmission protocols, since every lost packet would be transmitted again. This strategy is sub-optimal if packet losses are uncorrelated, as each retransmission is only useful to those receivers that have lost the given packet in the first place. It is likely that a single retransmission will only benefit a single receiver.

The impact of each retransmission can be maximized by using network coding [1], [2], [9], [5]. Researchers have shown that network coding can provide several advantages, namely improved throughput, robustness, security and lower complexity in communication networks [7], [3].

A. Network Coding

The basic operations performed in a network coding system are depicted in Figure 2. To lower the computational complexity of coding operations, large files or continuous streams are typically split into several equal sized chunks, also called *generations* [2], each consisting of g packets.

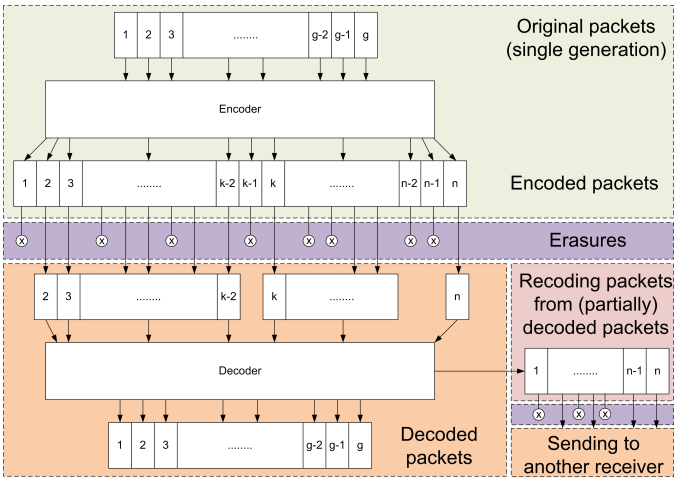


Fig. 2. Overview of Network Coding

The *encoder* (the top component in Figure 2) generates and transmits linear combinations of the original data packets in the current generation. Addition and multiplication are performed over a Galois field, therefore a linear combination of several packets will result in a packet having the same size as one of the original packets. With Random Linear Network Coding (RLNC), the coding coefficients are selected at random. Note that any number of encoded packets can be generated for a given generation. The middle layer represents the *wireless channel*, where packets are lost depending on the channel conditions. The received encoded packets are passed to the *decoder* (the bottom component in the figure), which will be able to reconstruct the original data packets after receiving at least g linearly independent packets.

The receiver nodes are also allowed to generate and send new encoded packets, even before decoding the entire generation. They form new linear combinations of the packets that they have previously received. This operation is known as *recoding*, and it is a unique feature of network coding. Traditional coding schemes require the original data to be fully decoded before it can be encoded again.

Another advantage of network coding is that it makes *"perfect coordination"* possible, where an arbitrary number of sending nodes can be used to serve the same generation to a receiver. Moreover, a receiver is no longer required to gather all data packets one-by-one, it can simply "hold a bucket" for a generation until it is full, that is enough linearly independent encoded packets are received. With RLNC, the randomly generated coding coefficient vectors from different senders are linearly independent with high probability (depending on which Galois field is used). Consequently, there is only a minimal need for signaling among the cooperating nodes.

III. Protocol Design

Our goal is to devise a protocol that minimizes the number of packets exchanged among the mobile clients, whilst maximizes the number of packet errors recovered. Moreover, we aim to repair the packet losses on-the-fly (as the data is being received), which imposes real-time constraints on the protocol. We call our protocol ECP (ENOC Cooperation Protocol).

We assume that the primary network is an LTE network, which uses systematic Raptor coding for broadcast transmissions. Thus the raw symbols (uncoded data packets) are transmitted first, and they are followed by several encoded packets to repair the losses. We intend to devise a best-effort protocol that uses the secondary network established among the receivers to conceal a significant part of these losses from the LTE Raptor decoder. It is important to note that ECP is not intended to provide full reliability, which remains the responsibility of the Raptor decoder. We assume that the protocol has read access to the raw symbols received on the primary network, and it can also write back newly recovered symbols to the Raptor decoder buffer, thereby significantly lowering the perceived packet error rate and the required overhead.

A. Protocol operation

ECP is based on the principles of network coding, since the receivers cooperate by sending encoded or recoded messages, which contain information that is most likely innovative for all peers. The basic unit of operation is a generation, meaning that the cooperative cluster is only trying to fix a specific generation at any given time. The network nodes may form a new cluster for the next generation.

After receiving a certain generation on the primary network, a receiver can broadcast a NACK (Negative Acknowledgment) message on the secondary network if it has experienced any packet losses. This is a retroactive trigger mechanism in ECP. We apply semi-random back-off intervals to prevent multiple nodes from broadcasting NACK messages at the same time.

The back-offs are chosen so that it is more likely that the worst receiver sends out the first NACK. These messages contain information about how many packets were lost on the receiver.

When the other devices within range receive this packet, they will suppress their own NACKs, and in response they schedule several encoded data packets to be sent at a specific speed. The devices generate and broadcast encoded packets, which also convey information about their own packet losses.

Since RLNC is used to achieve perfect coordination, encoded/recoded packets from any of the nodes can be equally useful. Senders do not have to pay attention to select specific packets for specific receivers.

Consequently, the most essential question here is how many packets the devices should schedule and transmit in response to a NACK message. They can simply broadcast as many as the worst receiver needs. The nodes constantly gain information about the others' knowledge with every encoded packet they receive. These updates can be used to continuously adjust the remaining number of packets to be sent.

This simple approach however leads to sub-optimal performance in most cases. If there are 3 or more cooperating devices, then the task of "filling up" the worst receiver should be equally divided among the others. For example, suppose that there are 4 devices and the worst one needs 15 packets, then the other 3 can send 5 packets each.

If we have only 2 cooperating nodes, then it is very likely that they have some common erasures. Consequently, full recovery is often not possible in this case, and the devices should send less packets than the other one has lost. Specific information about the individual packet losses is necessary to determine the combined knowledge of the cooperative cluster. We include a short bitvector (called the knowledge vector) in all protocol messages that explicitly indicates which packets of the current generation are currently available on the sender of the message. The network nodes can quickly calculate the combined knowledge of the cluster by bitwise OR-ing these vectors from all receivers. To determine how many innovative packets can be sent to a given node, we can take the difference of the combined knowledge and the node's latest knowledge vector.

This improved approach can yield near optimal performance under ideal channel conditions and slow transfer rates. However, as we increase the transfer rate, we observe that information about the other nodes can quickly become out-of-date thus inaccurate. This may lead to the premature termination of the repair session, i.e. the nodes may schedule less packets than necessary. Therefore we allow the receivers to send secondary NACK messages for the current generation when the others have stopped sending, but the node is still unable to decode some of the received messages.

B. Implementation

Based on the protocol design ideas outlined above, we have implemented a prototype application using the Qt framework so that we can test our protocol on any Nokia phone, desktop computer or laptop.

This demo application emulates an incoming LTE packet flow on all devices synchronously. When the application starts, it enumerates its network interfaces in order to determine its IP and broadcast addresses corresponding to the WLAN network interface. One device is used to start the simulation by broadcasting a command packet that contains all the simulation parameters. The other devices receive this packet, extract the parameters, and initiate the packet flow simulation. They all use the same random seed to generate random data packets deterministically, as if these were received on the LTE network. Packets are generated continuously with the same data rate on all clients. The devices also drop certain packets at random to simulate packet losses. Note that they do this independently from each other. Thereby all devices possess some parts of the incoming datastream, and they also have some "holes" that ECP should be able to fill on-the-fly.

We consider a single Raptor code source block, e.g. 1024 packets in a simulation. The entire block can be segmented into smaller generations (e.g. 64 packets) that are sufficiently small for network coding calculations on resource-constrained mobile devices. ECP works on these generations as depicted in Figure 3.

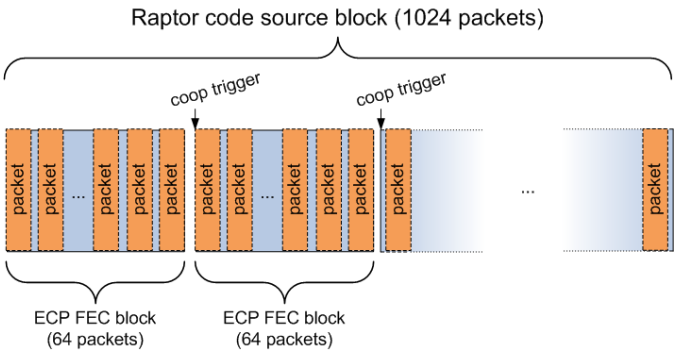


Fig. 3. Mapping between Raptor code source blocks and ECP generations.

When the first 64 packets of the emulated packet flow are (partially) received, the ECP trigger mechanism is activated, and the nodes generate a random back-off interval before sending a NACK (Negative Acknowledgment) message. When they receive a NACK, the devices generate and broadcast several encoded packets with a specified data rate in order to repair the packet losses in the cluster.

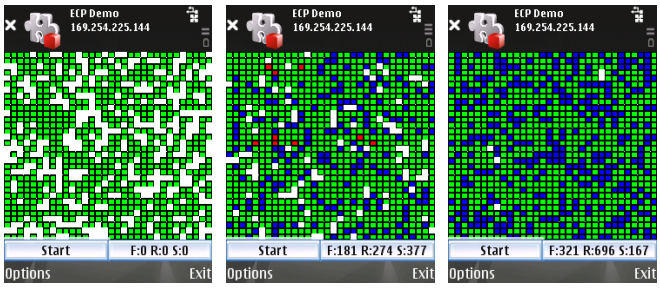
Network coding involves a computational overhead which might be prohibitive from a practical point of view. The authors in [5] proposed an efficient solution for mobile devices, namely to use the binary Galois field, GF(2) to simplify all calculations. Since this approach increases the probability of generating linearly dependent (i.e. useless) encoded packets, we have also implemented arithmetics over another finite field, GF(2^8), which is more computationally intensive, but almost totally free of linearly dependent packets. For a given simulation, all devices use either GF(2) or GF(2^8), as dictated by the source node in the simulation parameters.

The application can run simulations with different data rates, which is quite useful when we examine the implications of time constraints on the protocol performance. Moreover, data rates can be increased automatically for stress tests, and the collected simulation results from all participating devices can be transmitted to a logging server.

C. Visualization

The application also has a simple visualization grid with colors, hence we can observe as the packet losses are being repaired by the neighboring devices. In Figure 4 we show screenshots of this visualization grid after finishing simulations with 1, 2 and 6 devices.

A green box signifies a packet that was received on the simulated primary network. A red box signifies an encoded packet that was received from another device, but it has not been decoded yet, thus it can be considered ”dirty data”. A blue box signifies a packet that was originally lost on the primary network, but it has been recovered by ECP. Obviously, our objective is to maximize the number of blue boxes and to eliminate all red boxes.



(a) Single device (b) One of 2 devices (c) One of 6 devices

Fig. 4. Visualization grid after complete simulations with 1024 packets and 30% simulated packet loss.

IV. Measurements

We have assembled a testbed to demonstrate the capabilities of our protocol, and to refine its design. The cooperative cluster consists of 6 Nokia N95 (or N95 8GB version) mobile phones that run the demo application described above. These devices form an ad hoc WLAN network for short-range communication. All ECP packets are transmitted on this network.

It is important to design ECP so that it can work well with a various number of neighboring devices. In this testbed we can test its operation with 1-6 devices. With one device, the objective is to minimize outgoing traffic, although we have to make sure that the devices can detect each other’s presence and switch on cooperation when necessary. With 6 devices, we also aim to send as few packets as possible to minimize the overall energy consumption.

A. Performance evaluation

The main purpose of this testbed is to evaluate the performance of ECP on actual mobile devices, since it is a vital step in protocol design to gain feedback from real networks.

Our first performance metric is the required overhead to complete the transmission, as measured on the primary source node (i.e. the LTE base station).

In Figure 4 we can easily count the white ”holes”, each of those would require a retransmission from the source. In these simulations we used a source block consisting of 1024 packets with 30% simulated packet loss. We expect around 300 lost packets if a device is alone, that is why we see around 300 holes in the left screenshot (4a) after a finished simulation.

The middle screenshot (4b) illustrates the cooperation gain with just 2 devices. In this case we were able to fill 70% of the holes, that means a 70% reduction in the required overhead. Some packets were not fully decoded here (red boxes), which indicates sub-optimal performance with 2 devices.

The right screenshot (4c) shows the end result of a simulation with 6 devices, where 98% of the holes were filled. Therefore only a minimal amount overhead is required from the base station in this case. There are no red boxes here, and the remaining holes are actually correlated erasures, i.e. packets that were lost on all 6 devices of the cluster.

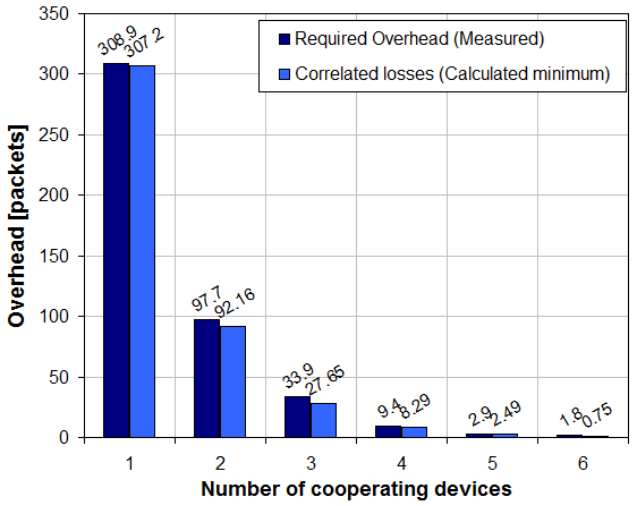


Fig. 5. Required overhead on the primary network.

Figure 5 shows the average required overhead as the number of cooperating devices increases. This graph is based on several hundred measurements performed with GF(2). As we can observe, the cooperation gain is quite significant even with 2 devices. Note that packets, that were lost on all devices, cannot be recovered. Their number is shown as correlated losses in Figure 5. In most cases ECP is able to achieve very close to the maximum cooperation gain, i.e. it can repair almost all packet losses that can be repaired by the cluster.

Our second performance metric is the number of packets exchanged on the WLAN network. The overall energy consumption of the system is heavily influenced by this measure. In the testbed, we can measure the number of transmissions (packets sent and received) on a single device, and we can aggregate these values from all devices for a given simulation.

In our previous work [17] the theoretical upper and lower bounds for this measure were established considering an ideal system based on network coding. The upper bound can be calculated by doubling the expected number of recoverable (i.e. non-correlated) packet losses on a single device. In the worst case, the device has to both send and receive that amount of packets. As we have mentioned before, the sending part can be optimized if there are more than two devices. We can divide the number of packets to be sent among all nodes. This principle is used to establish a lower bound, which is calculated for the worst receiver (on the primary network). For this device, the expected number of recoverable packet losses is higher than the average, and this is the amount of packets that it has to receive. It should also send its fair share of the expected number of recoverable losses on the second worst receiver. The lower bound is obtained by adding the number of packets received and sent. This is based on the observation that any packet exchanged on the WLAN network is either sent or received by a given node.

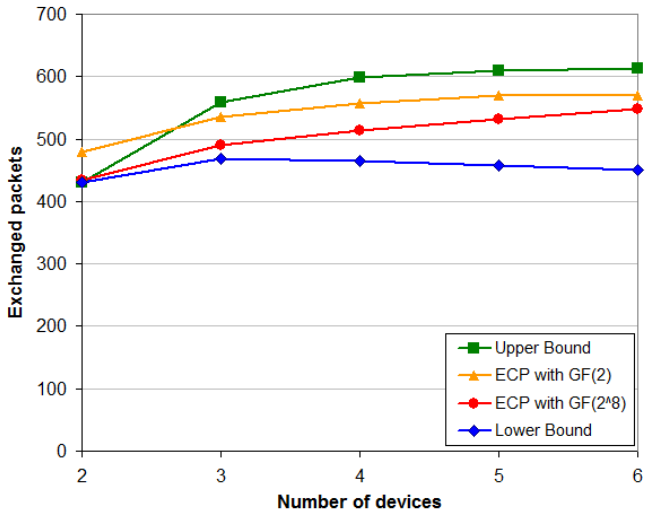
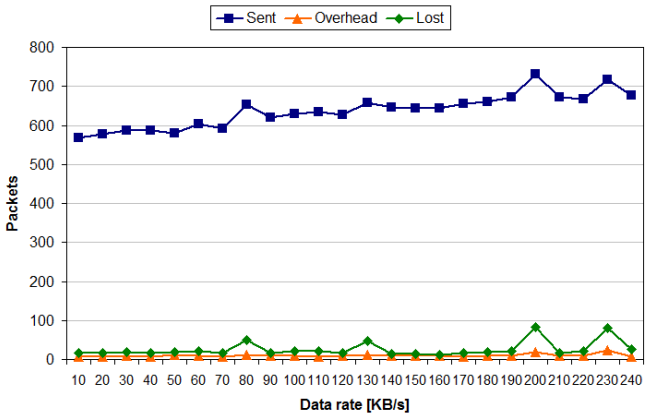


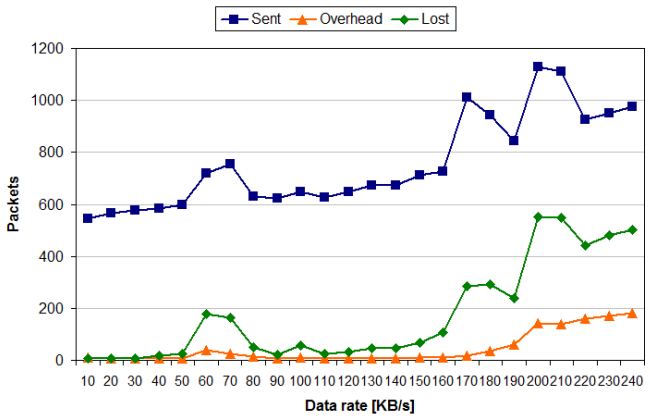
Fig. 6. Number of exchanged packets on the WLAN network as a function of the cluster size

The lower and upper bounds are plotted in Figure 6 together with the actual numbers measured on the testbed using GF(2) and GF(2^8). The performance with GF(2^8) is quite close to the lower bound for 2 and 3 devices, but it is getting worse for larger cluster sizes. On the other hand, using GF(2) results in generating some linearly dependent (i.e. useless) packets, which account for the performance gap between the two Galois fields. For 6 devices, GF(2) performs similarly to GF(2^8), which is an interesting result. It can be explained by the fact that the processing delay is significantly smaller for GF(2), therefore the protocol decisions (how many packets to send) are based on more up-to-date information.

The third performance metric is the compliance with real-time constraints. This can be very important for video playback



(a) ECP with GF(2)



(b) ECP with GF(2^8)

Fig. 7. Performance at different data rates with 4 devices

or similar services, where the packet losses should be repaired on-the-fly, i.e. faster than the native playout data rate of the video stream. In the testbed, we can adjust the data rate of the simulated LTE packet flow, and we can measure how the increased data rates influence the protocol performance. In Figure 7 we have plotted the number of total packets sent, the required overhead and the number of lost packets as measured in consecutive simulations with 4 devices with increasing data rates ranging from 10 to 240 KB/s. As we can see in Figure 7a, these performance indicators were almost unaffected when we used GF(2). We can only notice a slight increase in the number of sent packets. However, if we look at Figure 7b, we observe that the situation is much worse for GF(2^8). Encoding and decoding operations over GF(2^8) impose a heavy load on the CPU, which leads to frequent packet losses on the WLAN network. Consequently, the devices will try to send more packets to counteract these losses, and the number of sent packets increases almost twofold. As a result, the protocol is no longer able to recover all the original losses with data rates higher than 180 KB/s. For higher data rates, GF(2) performs better than GF(2^8) due to the high computational overhead associated with GF(2^8). Note that the highest data rate here,

240 KB/s corresponds to around 2 Mbit/s, which is more than sufficient for full quality videos on state-of-the-art mobile devices.

V. Future works

Before this protocol can be deployed in real-life mobile networks, we need to investigate the effects of peer mobility and dynamic behavior. Mobile devices may join and leave the cluster at any time, and typically they are also moving around, therefore static cooperative clusters are not to be expected in reality. Extensive measurements are needed to analyze the protocol performance in dynamic settings.

ECP was designed so that it can work well with multiple disjoint clusters if certain nodes move from one to another. Basically a new cluster is formed for every generation (e.g. 64 packets). If a cluster stays together for that short period of time, near-optimal performance can be achieved, and the next generation is handled by a new (possibly different) cluster.

The protocol can also be used for coverage extension in multi-hop wireless networks to provide services on mobile phones that are outside the range of 3G, 4G or LTE networks. If some nodes are *not* directly reachable by the source, then it is still possible to deliver the data stream to these receivers with the help of relaying nodes that propagate the received data farther away from the source. As it was shown in [16], the fundamental problem of data dissemination in multi-hop networks is the dynamic selection of relays and finding the proper scheduling scheme for the packet flow.

VI. Conclusion

In this paper we have introduced a protocol to facilitate the dissemination of multimedia content using cooperative clusters of mobile devices. We considered a scenario where these devices receive data from a broadcast transmission on a primary network (e.g. LTE network), and they use a secondary network (e.g. ad-hoc WLAN network) to recover packets that are lost during the transmission. We proposed a solution based on random linear network coding to facilitate the efficient exchange of information in the cooperative cluster. A demo application running on Nokia phones has been presented to show the feasibility of this approach. We observed that in most cases the protocol is able to realize almost the maximum cooperation gain, that is to recover all packets which are lost on the primary network. Meanwhile, the number of packets exchanged on the secondary network was kept close to the minimum. Moreover, we showed that packet errors can be recovered on-the-fly with data rates as high as 2 Mbit/s.

Acknowledgments

This work was partially financed by the CONE project (Grant No. 09-066549/FTP) granted by Danish Ministry of Science, Technology and Innovation as well as by the collaboration with Nokia/Renesas, Oulu throughout the ENOC project.

References

[1] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, Jul 2000.

[2] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. Network coding: an instant primer. *SIGCOMM Comput. Commun. Rev.*, 36(1):63–68, 2006.

[3] Christina Fragouli and Emina Soljanin. *Network Coding Applications*. Now Publishers Inc, January 2008.

[4] Z.J. Haas and T.C. Chen. Cluster-based cooperative communication with network coding in wireless networks. In *The 2010 Military Communications Conference - Unclassified Program*, pages 596–603, San Jose, California, USA, October 2010.

[5] J. Heide, M. Pedersen, F.H.P. Fitzek, and T. Larsen. Network coding for mobile devices - systematic binary random rateless codes. In *Workshop on Cooperative Mobile Networks 2009 - ICC09*. IEEE, June 2009.

[6] J. Heide, M. Pedersen, F.H.P. Fitzek, T. Madsen, and T. Larsen. Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast. In *4th International Mobile Multimedia Communications Conference (MobiMedia 2008)*, Oulu, Finland, July 2008. ICTS/ACM.

[7] Tracey Ho and Desmond Lun. *Network Coding: An Introduction*. Cambridge University Press, 2008.

[8] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36(4):243–254, 2006.

[9] D.E. Lucani, F.H.P. Fitzek, M. Medard, and M. Stojanovic. Network coding for data dissemination: It is not what you know, but what your neighbors know. In *RAWNET/WNC3 2009*, June 2009.

[10] Ryutaroh Matsumoto. Construction algorithm for network error-correcting codes attaining the singleton bound. *VOL.E 90-A*, 9:1729, 2007.

[11] L. Militano, F.H.P. Fitzek, A. Iera, and A. Molinaro. A genetic algorithm for source election in cooperative clusters implementing network coding. In *IEEE International Conference on Communications (ICC 2010) - CoCoNet Workshop*, May 2010.

[12] Joon-Sang Park, M. Gerla, D.S. Lun, Yunjung Yi, and M. Medard. Codecast: a network-coding-based ad hoc multicast protocol. *Wireless Communications, IEEE*, 13(5):76–81, October 2006.

[13] M.V. Pedersen, F.H.P. Fitzek, and Torben Larsen. Implementation and performance evaluation of network coding for cooperative mobile devices. *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, pages 91–96, May 2008.

[14] M.V. Pedersen, J. Heide, F.H.P.Fitzek, and T. Larsen. Pictureviewer - a mobile application using network coding. In *European Wireless 2009*, Aalborg, Denmark, May 2009.

[15] H. Shojania and B. Li. Random network coding on the iphone: Fact or fiction? In *ACM NOSSDAV 2009*, June 2009.

[16] P. Vingelmann, F.H.P. Fitzek, and D. E. Lucani. Application-level data dissemination in multi-hop wireless networks. In *IEEE International Conference on Communications (ICC 2010) - CoCoNet Workshop*, May 2010.

[17] Q. Zhang, J. Heide, M. Pedersen, F.H.P. Fitzek, Jorma Lilleberg, and Kari Rikkinen. Network Coding and User Cooperation for Streaming and Download Services in LTE Networks. In *Network Coding: Fundamentals and Applications*. Academic Press, 2011.